INTEGRATING CONTROL APPLICATIONS INTO DIFFERENT CONTROL SYSTEMS∗

M. Killenberg†, M. Hierholzer, C. Schmidt, DESY, Hamburg, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany
J. Wychowaniak, Łódź University of Technology, Łódź, Poland

Abstract

Porting complex device servers from one control system to another is often a major effort due to the strong code coupling of the business logic to control system data structures. Together with its partners from the Helmholtz Association and from industry, DESY is developing a control system adapter. It allows writing applications in a control system independent way, while still being able to update the process variables and react on control system triggers. We report on the status of the project and the experience we gained trying to write portable device servers.

INTRODUCTION

With embedded systems becoming more and more powerful, the algorithms in the devices which are accessed via control systems are becoming more and more advanced. Especially on MicroTCA [1] systems the hardware usually features a powerful multi-core CPU with several GB of RAM. The MicroTCA.4 [2] extension brings trigger and clock lines, as well as large rear transition modules which can be used for demanding analogue control applications.

Many particle accelerators which are currently being built are using or will use MicroTCA.4 for control of the radio frequency (RF) in the accelerator, for instance FLASH [3] and the European XFEL [4] hosted at DESY, Hamburg, ELBE [5] at Helmholtz-Zentrum Dresden-Rossendorf, or FLUTE [6] at KIT, Karlsruhe. The complex RF control applications shall be reused across the different accelerators, while all the facilities are using different control systems. It turned out that porting the software to a different control system is a major effort because the code is strongly coupled to the original control system. This lead to the idea to have an adapter layer between the device library, which implements the algorithms, and the control system, which provides the communication protocols and integration into the facility’s control infrastructure. This adapter shall be part of the MicroTCA.4 User Tool Kit (MTCA4U) [7, 8], a collection of libraries to facilitate the implementation of control applications.

REQUIREMENTS

The main task of the adapter is to allow application code to access process variables which are communicated to the outside world in a control system independent way. For this, the adapter has to use the functionality which is provided by the control system, like communication protocols or the addressing scheme.

The part of code which is device and control system dependent has to be minimal, zero if possible. This type of code is causing the huge workload when porting and maintaining applications for multiple control systems.

Abstraction is easy to achieve if data is simply copied back and forth between two domains, but this comes with a performance penalty. So an additional requirement to the adapter is to avoid unnecessary copying, especially of large data structures like arrays. In addition, the adapter should be thread safe and usable in real time application.

As a starting point and to check if the abstraction is working, the adapter is tested with two control systems: DOOCS [9] and EPICS [10]. DOOCS (used at DESY for FLASH and the European XFEL), has an object-oriented data model written in C++. EPICS 3, one the most widely used control systems for particle accelerators and used at FLUTE, has a channel-based C API. We intentionally used two conceptually different control systems, hoping that the abstraction needed to work with these two should also allow other control systems to be used without too much modifications in the design.

DESIGN CONCEPT

The first implementation of the adapter focuses on process variables. It provides data structures for scalars (8, 16 and 32 bit signed and unsigned integers, single and double precision floating point), strings, and arrays of the numerical data types. Each process variable is identified by a unique name which describes its function inside the device code (“TEMPERATURE” for instance for a device with a temperature sensor). The name does not contain information where the device is installed and in which context it is used. This part depends on the control system and the facility, and is added in the control system specific part of the adapter, not in the device part.

The original idea to avoid copying, especially for large arrays, was to have a single instance of the data. This would be stored in a control system dependent type, an instance which always has to be there to work with the particular control system. The adapter would provide a wrapper, which would be used inside the business logic. But it turned out that this approach is not viable. DOOCS and EPICS have different locking schemes for their variables, and only the control system side could know when it is safe to access


Copyright © 2015 CC-BY-3.0 and by the respective authors

Proceedings of ICALEPCS2015, Melbourne, Australia TUD3O05

Integrating Complex or Diverse Systems
the variables, but not the control system independent device part.

The solution is to have a control system independent instance on the device side which can be accessed at any time, and the control system variable on the other side, which can have a control system lock (Fig. 1). These variables have to be synchronised, which means that the abstraction at this point requires one copy which cannot be avoided. As not all control systems allow a variables to be input and output at the same time, it was decided to restrict process variables to be unidirectional (“control system to device” or “device to control system”).

The data being transferred is always the reference to a buffer, not the buffer itself, which avoids unnecessary copying of large data structures. When receiving, the receiver will pop the head of the “filled buffers” queue. If it could get (the reference to) a new buffer, it will push the now outdated buffer to the “available buffers” queue, so the sender can reuse it. In case there are no updated buffers available, the receiver will hold on to the current buffer, as it contains the most up-to-date information that is available on the receiver side.

Before actually sending, the sender will pop the head of the available buffers queue to be sure it has a new buffer which it can fill after sending (at all times there must be at least one buffer on each the sender and the receiver side). After that, the buffer to be send is pushed to the “filled buffers” queue. Both receiver and sender first pop the head of the queue where they retrieve the next buffer, and then push the buffer which has been processed for use by the other side. As this can happen at the same time, it means there have to be at least four buffers.

In contrast to a triple buffer, which is a common scheme in real time applications, this approach does not require a dirty flag for the buffer, and the receiver does not have to swap back to keep the latest available buffer in case no updated buffer is found. As a further advantage, the number of buffers can simply be increased, allowing a longer queue in case there are fluctuations on the receiver side, but it is fast enough to catch up with a certain backlog.

If the receiver is too slow to process data at the rate at which the sender is producing it, data will be lost. This cannot be avoided. In our implementation with a fixed number of buffers it means the “available buffers” queue is empty. In a naive implementation, the sender would keep its current buffer to overwrite it (buffer 3 in Fig. 3). However, it is not a good solution to stop sending because this would result

---

1 A copy can be avoided if the control system type allows swapping of the internal buffer.

2 For scalars copying the value is not more expensive than copying the pointer. In this case the values are directly copied and no additional buffers are needed.
The device configuration files for the control system had to be written in addition.

As “proof of concept”, some example devices with a couple of scalars and arrays have been created and tested using either DOOCS or EPICS, and have afterwards been ported to the other control system. This porting worked smoothly in both directions. As expected, the device code itself did not have to be modified. The amount of device-specific code is very small, on the control system side, the amount of device-specific code is down to three lines of device-dependent control system code, for DOOCS as well as for EPICS, independent from the number of process variables.  

To make use of additional control system features like variable limits or histories, these currently have to be implemented as device-specific code on the control system side. This introduces additional work when porting and maintaining the code for multiple control systems. It is currently being discussed how the adapter can be extended to allow the definition of the features on the device side, but use the functionality from the control system. Especially features which are expensive to implement like archiving and history should not be duplicated in the adapter.

**CONCLUSIONS**

The MTCA4U control system adapter provides an interface to use process variables in a device library without introducing a coupling to a particular control system. The implementation is lock free and transfers pre-allocated buffers without copying. This step was necessary to allow operation in control systems with different locking mechanisms. It also enables the device logic to use process variables in a real time thread.

The adapter consists of a common part, which implements the decoupling, and a control system specific part, which provides the particular bindings. As a proof of concept, adapters for DOOCS and EPICS have been implemented. An extension for OPC-UA is planned. As next steps, the abstraction of additional control system features like variable limits, engineering units and histories are intended.

All software is published under the GNU General Public License and available in the respective software repositories [11–13].

**REFERENCES**


---

3 The device configuration files for the control system had to be written in addition.
create("VOLTAGE", CS -> Dev)
create("TEMPERATURE", Dev -> CS)

Figure 4: The creation of process variables is requested on the device side. The instantiation on the control system side is automated as much as possible inside the adapter.


[9] The Distributed Object Oriented Control System (DOOCS), http://doocs.desy.de/


